

```

public class BinarySearchTree<ELEMENT extends Comparable<ELEMENT>> extends
BinaryTree<ELEMENT> {

    public BinarySearchTree() {
        super();
    }

    public void add(ELEMENT item) {
        if (this.root == null) {
            this.root = new BTNode<ELEMENT>(item, null, null);
        } else {
            BTNode<ELEMENT> temp = this.root;
            BTNode<ELEMENT> prev = null;
            while (temp != null) {
                prev = temp;
                if (item.compareTo(temp.item) < 0) {
                    temp = temp.left;
                } else {
                    temp = temp.right;
                }
            }
            temp = new BTNode<ELEMENT>(item, null, null);
            if (item.compareTo(prev.item) < 0) {
                prev.left = temp;
            } else {
                prev.right = temp;
            }
        }
    }

    public ELEMENT remove(ELEMENT item) {
        return removeByCopy(item);
        //return removeByFusion(item);
    }

    private ELEMENT removeByCopy(ELEMENT item) {
        BTNode<ELEMENT> find = this.root;
        BTNode<ELEMENT> prev = null;
        while ((find != null) && (find.item.compareTo(item) != 0)) {
            prev = find;
            if (item.compareTo(find.item) < 0) {
                find = find.left;
            } else {
                find = find.right;
            }
        }
        if (find == null) {
            return null;
        }
        if (find.left == null) {
            if (prev == null) {
                this.root = find.right;
            } else {
                prev.right = find.right;
            }
        } else if (find.right == null) {
            if (prev == null) {
                this.root = find.left;
            } else {
                prev.right = find.left;
            }
        } else {
            BTNode<ELEMENT> successor = find.right;
            BTNode<ELEMENT> successorPrev = find;
            while (successor.left != null) {
                successorPrev = successor;
                successor = successor.left;
            }
            if (prev == null) {
                this.root = successor;
            } else {
                prev.right = successor;
            }
            if (successor == find.right) {
                successor.left = find.left;
            } else {
                successor.left = find.left;
                successor.right = find.right;
            }
        }
        return find.item;
    }
}

```

```

        }
    }

    if (find == null) {
        throw new RuntimeException("No existe el elemento o el árbol está vacío");
    } // find es el nodo con el valor a extraer y prev el padre de ese nodo
    ELEMENT save = find.item;
    BTNode<ELEMENT> node = find;
    if (node.right == null) { // no hay subárbol derecho
        node = node.left; // nodo con un descendiente u hoja
    } else {
        if (node.left == null) { // no hay subárbol izquierdo
            node = node.right; // nodo con un descendiente u hoja
        } else { // dos descendientes
            BTNode<ELEMENT> last = node;
            BTNode<ELEMENT> temp = node.right; // a la derecha (mayores)
            while (temp.left != null) { // busca a la izquierda el menor
                last = temp;
                temp = temp.left;
            }
            // temp es el menor de los mayores
            node.item = temp.item; // hace la copia
            if (last == node) {
                last.right = temp.right;
            } else {
                last.left = temp.right;
            }
            temp.right = null;
        }
    }
    // reajustar el arbol
    if (find == this.root) {
        this.root = node;
    } else {
        if (prev.left == find) {
            prev.left = node;
        } else {
            prev.right = node;
        }
    }
    return save;
}

```

```

private ELEMENT removeByFusion(ELEMENT item) {
    BTNode<ELEMENT> find = this.root;
    BTNode<ELEMENT> prev = null;
    while ((find != null) && (find.item.compareTo(item) != 0)) {
        prev = find;
        if (item.compareTo(find.item) < 0) {

```

```
    find = find.left;
} else {
    find = find.right;
}
}
if (find == null) {
    throw new RuntimeException("No existe el elemento o el árbol está vacío");
}
ELEMENT save = find.item;
BTNode<ELEMENT> node = find;
if (node.right == null) {
    node = node.left;
} else {
    if (node.left == null) {
        node = node.right;
    } else {
        BTNode<ELEMENT> temp = node.right;
        while (temp.left != null) {
            temp = temp.left;
        }
        temp.left = node.left;
        node = node.right;
    }
}
if (find == this.root) {
    this.root = node;
} else {
    if (prev.left == find) {
        prev.left = node;
    } else {
        prev.right = node;
    }
}
find.left = find.right = null;
return save;
}
}
```